On the Use of Social Trajectory-based Clustering Methods for Public Transport Optimization

Jordi Nin, David Carrera, and Daniel Villatoro

[1] Barcelona Supercomputing Center (BSC)
Universitat Politècnica de Catalunya (BarcelonaTech)
Barcelona, Catalonia, Spain
nin@ac.upc.edu, dcarrera@ac.upc.edu
[2] Barcelona Digital Technology Centre, Barcelona, Catalonia, Spain
dvillatoro@bdigital.org

Abstract. Public transport optimisation is becoming everyday a more difficult and challenging task, because of the increasing number of transportation options as well as the exponential increase of users. Many research contributions about this issue have been recently published under the umbrella of the smart cities research. In this work, we sketch a possible framework to optimize the tourist bus in the city of Barcelona. Our framework will extract information from Twitter and other web services, such as Foursquare to infer not only the most visited places in Barcelona, but also the trajectories and routes that tourist follow. After that, instead of using complex geospatial or trajectory clustering methods, we propose to use simpler clustering techniques as k-means or DBScan but using a real sequence of symbols as a distance measure to incorporate in the clustering process the trajectory information.

Keywords: Smart Cities, Geospatial Clustering, Metric Spaces, OSA Distance, Cloud Computing, High Performance Computing

1 Introduction

Trajectory clustering algorithms [24] group similar trajectories into groups (clusters), thus discovering common trajectories. These methods are different from geospatial clustering [38]. This latter clustering methods group similar objects (points in an Euclidian or geodesic space) based on their distance, connectivity, or relative density in the space.

Since geospatial clustering methods are in general easier than trajectory clustering algorithms, they has been employed in the field of spatial analysis for years. Spatial clustering is the process of grouping similar objects based on their distance, connectivity, or relative density in space. Thanks to that, such methods prevail over trajectory clustering algorithms in pattern recognition smart cities applications, such as public transport optimization. The main problem behind the use of geospatial clustering methods for this purpose is that patterns (most of times trajectories) must be builded over geospatial clustering results. To do

that, expert knowledge is usually required making this approach less useful for big cities and highly dynamic city environments.

Since trajectory clustering algorithms include by definition all the trajectory information, it is possible to adjust these algorithms to reduce the expert's time to take the decision about with global trajectories are the most interesting ones. In addition, trajectory clustering algorithms can help the data practitioners to discover common sub-trajectories inside a cluster. This information can be very valuable in many applications, especially if we have regions of special interest for analysis inside a big city, such as, the city centre or certain regions with a high number of touristic attractions or city services.

All trajectory clustering algorithms are based on computing a distance assuming that trajectory elements are represented by means of space coordinates. This lack of element semantics makes that clusters construction only considers space similarities. In this paper, we would like to study how to convert trajectory coordinates to symbols, in such a way, we can include semantics inside the trajectory elements. Therefore, it will be possible to group similar trajectories considering the nature of their elements, for instance if they are touristic attractions, city services, restaurants, etc... The main drawback of this approach is that it is required to define an appropriate distance for this type of sequences.

In this paper we propose to use the Optimal Symbol Alignment (OSA) distance [18] for *semantic-aware* trajectory clustering. To do that, we would like to integrate it into the clustering framework ELKI [1] to perform some experiments using trajectories about tourists visiting Barcelona.

1.1 Paper Organization

The rest of this paper is organized as follows. Firstly, in Section 2 we introduce some basic concepts about sequences of symbols distances. We also provide a complete definition of the Edit and OSA distances. Then, in Section 3 we provide a taxonomy for the currently used clustering algorithms, as well as, some implementation decisions we have already taken. In Section 4 we describe how we have obtained the data required for our clustering analysis using several social network services. Later, in Section 5, we mention several computational issues we must consider due to the high amount of data available. Finally, Section 6 depicts the following steps in our proposal.

2 Distances for Sequences of Symbols

Comparison functions for sequences (of symbols) are important components of many applications, for example clustering, data cleansing and integration.

For this reason, there is a lot of work in computing similarities among sequences of symbols [8, 15, 29]. However, the similarity measures presented in most of those works either do not fulfil the mandatory conditions to be a real distance (most of the times, because the triangular inequality does not hold) or do not contain a proof for that. Formally, a distance function d must satisfy the following properties:

1. Symmetry: $d(A, B) = d(B, A)$ for all sequences A, B
2. Positivity: $d(A, B) \geq 0$ for all sequences A, B
3. Reflexivity: $d(A, A) = 0$ for all sequence A
4. Triangular Inequality: $d(A, B) \leq d(A, C) + d(B, C)$ for all sequences A, B, C

To the best of our knowledge, there are only three sequence measures that fulfil these conditions: the Hamming distance [17], the Levenshtein (Edit) distance [25] and the OSA Distance [18]. The remaining measures are similarity functions instead of real distances because they do not comply with the triangular inequality (or this is not proved). For this reason the application of such measures to the scenarios where having a metric space is a must, such as metric spaces [5], clustering [19] or k-nearest neighbors algorithms [6], becomes unfeasible from a theoretical point of view.

The Hamming and Edit distances present also some problems. For instance, the Hamming distance can only be applied to sequences of the same length, while the Edit distance has a large, both practical and theoretical, complexity ($O(n^2)$). For these reasons many similarity measures have been developed, albeit sacrificing some of the mandatory properties of a distance. For example, the Jaro-Winkler distance [21] is very efficient in terms of practical computational cost when the compared strings are not too large. Therefore, it saves execution time (when compared to Edit distance) in applications where there are many comparisons to be done.

2.1 Edit Distance

The Edit distance [25, 32] measures the difference between two sequences, given by the minimum number of edit operations needed to transform one sequence into the other. An edit operation can be either an insertion, deletion or substitution of a single symbol, although many variations exist in which the set of allowed operations is larger or more restricted. In some way, Edit distance assumes that the differences between two sequences are due to typos or spelling errors.

The Edit distance has found a large variety of applications in many scenarios and has achieved very good results [31]. However, the Edit distance has a large complexity: its computation using classical algorithms [36] based on dynamic programming has a complexity equal to $O(n^2)$, where n is the size of the shortest string. Other algorithms to compute the Edit distance exist [34, 4], having a lower complexity of $O(dn)$, for example, where d is the real Edit distance. Note that, when two completely different strings have to be compared, the complexity of these variants is the same as that of the classical algorithm.

2.2 Optimal Symbol Alignment (OSA) Distance

The intuition behind the Optimal Symbol Alignment (OSA) distance [18] is that strings are close if they have many common symbols, and in addition their common symbols are placed in similar positions, in the strings being compared.

Given a finite alphabet of symbols X, let $A = (a_1, \ldots, a_{n_A})$ and $B = (b_1, \ldots, b_{n_B})$ be two sequences of symbols, where $a_i, b_j \in X$, for $i = 1, \ldots, n_A$, $j = 1, \ldots, n_B$. For any sequence of symbols A, we define as $X_A \subseteq X$ the subset of symbols that appear in A; that is, $X_A = \{x \in X \text{ s.t. } \exists i \in \{1, 2, \ldots, n_A\}$ with $a_i = x\}$. For a symbol $x \in X_A$, we also define the subset of positions $A_x = \{i \in \{1, \ldots, n_A\} \text{ s.t. } a_i = x\}$.

We define the OSA distance $d(A, B)$ between the sequences A and B as

$$d(A, B) \;=\; \sum_{x \in X_A \cup X_B} d(x, A, B),$$

where the value $d(x, A, B)$ is defined as

$$d(x, A, B) = \begin{cases} |A_x| & \text{if } x \in X_A - X_B \\ |B_x| & \text{if } x \in X_B - X_A \\ f(x, A, B) & \text{if } x \in X_A \cap X_B \end{cases}$$

Finally, we have to define the value of $f(x, A, B)$, which is the contribution of the symbol x to the distance $d(A, B)$, when this symbol x is included in both sequences A and B. Let us assume without loss of generality that $|A_x| \leq |B_x|$. The idea is to select the subset of $|A_x|$ positions j, from the set B_x, which are globally closest to the set of $|A_x|$ positions in A_x. Namely, if $i_1 < i_2 < \ldots < i_{|A_x|}$ are the positions in A_x, then we select $|A_x|$ positions $j_1 < j_2 < \ldots < j_{|A_x|}$ in B_x minimizing the global distance $|i_1 - j_1| + \ldots + |i_{|A_x|} - j_{|A_x|}|$. We use notation $j_h = \mathtt{pj}(i_h, A, B)$, for $h = 1, \ldots, |A_x|$, to denote the position in B_x that optimally matches position $i_h \in A_x$. We say that j_h is the *projection* of position i_h from sequence A to sequence B. For completeness, we also use the symmetric notation $i_h = \mathtt{pj}(j_h, B, A)$.

Each of these common symbols $a_{i_h} = b_{j_h} = x$, for $h = 1, \ldots, |A_x|$, will contribute with $\frac{|i_h - j_h|}{n_{AB}}$ to the value $f(x, A, B)$, where $n_{AB} = \max\{n_A, n_B\}$. In this way, we ensure that these contributions are bounded by 1. The remaining $|B_x| - |A_x|$ symbols will be considered as non-common symbols, so each of them will contribute with a 1 to the global distance $d(A, B)$.

Taking all these facts into account, we finally have

$$f(x, A, B) \;=\; (|B_x| - |A_x|) \;+\; \frac{1}{n_{AB}} \sum_{i_h \in A_x} |i_h - \mathtt{pj}(i_h, A, B)| \;.$$

Depending on the differences between the two sequences to be compared (more or less repeated symbols, more or less transpositions, etc.) the OSA distance $d_{\text{OSA}}(A, B)$ will be more or less similar to the Edit distance $d_{\text{Edit}}(A, B)$. But in any case, they will not be very far, because it is easy to prove that $\frac{d_{\text{Edit}}(A,B)}{2} \leq d_{\text{OSA}}(A, B) \leq 2 \cdot d_{\text{Edit}}(A, B)$, for any two sequences A, B.

3 Clustering

Clustering is the task of relating similar elements in a dataset to build groups and create general representations (centroids). It is widely used in many fields

as data mining, machine learning, image analysis, etc. Since clustering is a very general concept, there are a great variety of algorithms that apply clustering, focusing, for example, on possible centroids, density, data structures, etc.

Clustering algorithms are divided into different different categories: space partitioning, also called top-down methods, hierarchical methods, known as bottom-up methods as well or Density-Based Clustering Methods. Now, we review three well-known clustering algorithms which illustrate these categories: k-means (a space partitioning method), the agglomerative hierarchical clustering and DBScan

k-means algorithm [26]. It is one of the most commonly used clustering techniques. It is an algorithm to cluster n objects into k partitions ($k < n$). K-means starts by partitioning randomly the input objects into k initial sets. Then, it calculates the centroid of each set. Following, it constructs a new partition by associating each object with the closest centroid. Finally, the centroids are recalculated for the new clusters. This algorithm is repeated until it convergences, i.e. there is no changes in its centroids. Figure 1 shows an example of k-means over a dataset of 100 gaussian random pairs.

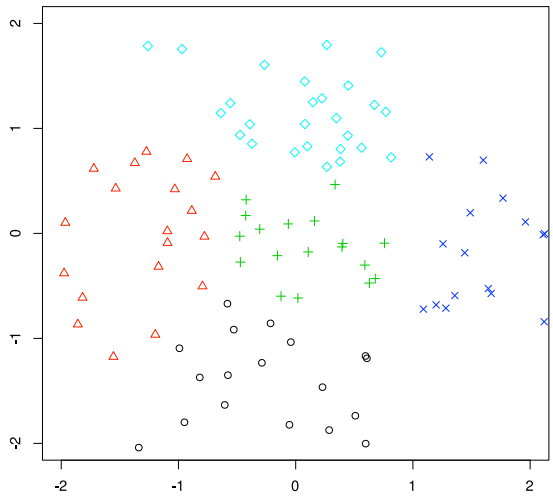

Fig. 1. Example of clustering using k-means algorithm with $k = 5$

Agglomerative hierarchical clustering [20]. This method builds a hierarchy tree, called dendrogram, from the individual elements by progressively merging clusters. Note that, at the beginning each element is considered as an independent cluster. The algorithm starts computing a distance matrix among all the elements to be clustered, where the distance in the (i, j) position corresponds to the distance between the i-th and j-th elements. Then, when clustering progresses, the

corresponding rows and columns have to be also merged. This algorithm does not explicitly builds a number of clusters, instead, we must decide the number of clusters and where we split them within the dendrogram. An example, over the same dataset as Figure 1, is illustrated by Figure 2, which suggests to cut at height 3 or 4.

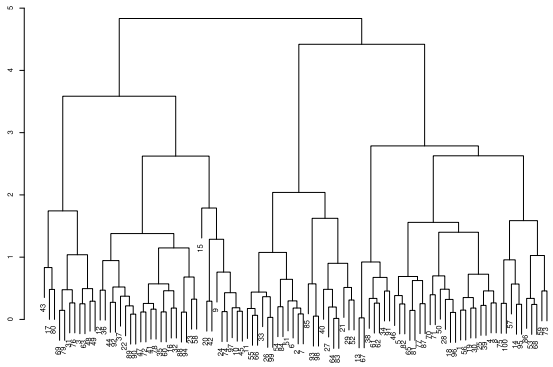

Fig. 2. Example of dendrogram after applying hierarchical clustering

DBScan [9]. It starts with an arbitrary starting point r that has not been visited. r's close neighbours are retrieved, and if this set contains sufficiently many points, a cluster is started. Otherwise, r is labeled as noise. Note that this point might later be found in a sufficiently sized environment of a different point and hence be made part of a cluster. If a point is found to be a dense part of a cluster, its neighbours are also part of that cluster. Hence, all points that are found within the neighborhood limit are added to the dense cluster. This process continues until the density-connected cluster is completely found. Then, a new unvisited point is retrieved and processed, leading to the discovery of a further cluster or noise. Figure 3 shows the output of DBScan within a two dense clusters dataset.

3.1 Geospatial Clustering

Geospatial clustering is a special kind of clustering, its main goal is to group similar objects based on their distance, connectivity, or relative density in space. General clustering methods can be used for geospatial clustering, however due to its inherent spatial nature, specific clustering algorithms categories have been defined in the last years:

Grid-based clustering methods. Such methods divide the clustering space into a finite number of cells and then perform all the clustering operations on the grid. Cells containing more than a certain number of points are considered to be dense. Contiguous dense cells are connected to form clusters. One examples of grid-based clustering methods is CLIQUE [2].

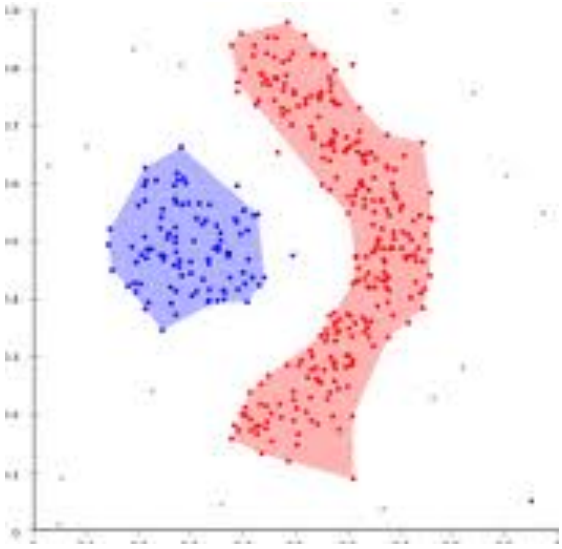

Fig. 3. Example of DBScan with 2 clusters

Constraint-Based Clustering Methods. These clustering methods add spatial constraints, such as obstacles, to obtain more desirable clusters for real geospatial information systems. Depending on the nature of the constraints and applications, constraint-based clustering methods includes four different types of constrains: constraints on individual objects, obstacle objects as constraints, clustering parameters as constraints, and constraints imposed on each individual cluster. [33]. DBRS+ [37] is one example of these clustering methods.

3.2 Trajectory Clustering

A number of trajectory clustering methods have been proposed. A common characteristic of first trajectory clustering methods is that they use the shapes of whole trajectories to do the clusters, using for instance hidden Markov models (HMM). Later, more complex clustering algorithms [23] where authors divide the space into a grid to allow to compute sub-trajectories and consider more complex patterns than using only the whole trajectory.

Our approach is far from these ideas. Here we would like to adapt classical clustering methods, which has been largely studied to cluster trajectories using well-defined distances for sequences of symbols.

3.3 Clustering Tools

Nowadays, it is possible to find a large variety of clustering tools, such as WEKA [16] . However, most of these tools are not able to work with big data or cannot be parametrized using an external distance.

To solve the first problem, MongoDB [28] offers special data structures and indices called Geohash [27]. When you create a geospatial index on legacy coordinate pairs, MongoDB computes geohash values for the coordinate pairs within the specified location range and then indexes the geohash values. Geohash values, recursively divide a two-dimensional map into quadrants until a desired level of specificity is reach. We have used MongoDB and geohash data structures to create our trajectory dataset.

In order to easily execute several clustering algorithms coming from the different families described in this section, we will use the ELKI framework [1]. ELKI separates data mining algorithms and data management tasks to allow for including different components inside the library. Additionally, ELKI allows for the use of external distances as the OSA distance described in Section 2.2. For these reasons we have selected ELKi for our experiments.

4 Social Sensing Trajectory Data Extraction

In order to capture citizens' mobility traces of citizens, different data sources have been used in the literature such as GPS traces ([39]), call detail records (CDR) from mobile phones ([7]), and even geopositioned social media ([30]).

In this work we focus on geopositioned tweets. Twitter allows developers to obtain all the geopositioned tweets within a certain bounding box that covers a certain city. We have used the data-acquisition architecture depicted in [35]. The dataset generated contains the geopositioned tweets generated in Barcelona during six months from July 2012 to the end of December 2012, resulting in approximately 1.160.000 geopositioned data traces.

To transport the geopositioned tweets into actual trajectories, we borrow a methodology presented elsewhere ([12]): a trajectory can be understood as the set described by at least two consecutive tweets with a minimum distance of 100m amongst them and published in less that 75 minutes of difference.

Moreover, we also profit from a semantically-enhanced module that allows to detect the origin country of each user leaving a digital mobility trace. Each tweet contains the user-specified origin location, allowing users to specify free text values. Other than handling fake values, such as "From heaven", user-specified locations can be referred at different levels of granularity (GPS coordinate, neighborhood, city, region, or country level) and in different languages (e.g. *London, Londra, Londres*, etc...). In order to deal with this ambiguity, we use the GeoNames service [13], which provides an unique country identifier per any input, being also succesful dealing with fake values [3].

Therefore, our technological infrastructure allows us to capture user trajectories and classify them per user origin country.

By applying clustering algorithms on the digital traces of our 6 months dataset, we obtain clusters of activity whose cluster identifier provides us with an unique tag to identify users origins and destinations.

[3] A human-supervised test on 100 random user-specified locations obtained from our dataset obtained an 72% accuracy rate.

5 High Performance Computing Challenges in Geospatial Clustering

To avoid the problems associated to persisting data to disks, persistent memories are leveraged (Flash memory at the current time, any kind of Storage Class Memories in the near future).

To increase the address space made available to a particular workload, the memory of all nodes involved in the execution of such workload are given access to the memory of all other compute nodes thorugh software APIs.

The major problem from the infrastructure point of view for Geospatial Clustering is dealing with large amounts of data. It is a common case that the performance of clustering algorithms is I/O bound becuase they are data-intensive and process very large data sets. Therefore, novel ways of managing data are required to deliver scalability and performance. The common approach nowadays is to distribute data across many compute nodes and provide applications with a flat and shared name space to access data, independently if it is stored locally or remotely.

The single namespace can be provided either by a distributed file system (the case of MapReduce Distributed File System [3, 14]) or using key/value pairs (the case for most NoSQL databases [22, 28] and key/value stores [11]).

Over the last years an intersection of different technologies is gaining momentum performance-wise: RDMA-enabled network technologies and persistent memories. The goal of combining these technologies is to provide low latency and high bandwidth all-to-all in a network topology, and therefore fast access to all data in a distributed dataset.

High speed networks require lightweight protocol stacks and CPU offloading to move data between nodes at high speeds (e.g. using Infiniband verbs instead of a heavyweight TCP/IP stack to achieve 56Gbps bandwidth in Infiniband FDR networks), what is achieved using RDMA-enabled networks such as Infiniband or iWARP.

At the same time, to achieve high bandwidth to store data, fast memories are used instead of slow rotational disks. Such memory can be accessed in different ways: through conventional disk interfaces, what is the case of Solid State Disks (SSD); through conventional PCIe buses, what is the case of PCIe Flash boards; or directly through the memory buses in the processors, what is the case of Phase Change Memories (PCM) or any generic Storage Class Memory (SCM) technology.

The outcome of this technology trend is middlewares that allow for transparent access to remote or local data at the same speeds and with the same latencies, unifying the memory space of all nodes involved in the execution of a workload, and simplifying the programmability of the applications by providing simple user APIs, such as the Blue Gene Active Storage [10] (BGAS) platform does.

6 Conclusions

In this abstract we have described all the components of a possible framework for public transport optimisation in big cities, in our case Barcelona. We have described several distances and clustering algorithms to illustrate that combining a distance for sequences of symbols with classical clustering algorithms is possible to create a clustering algorithm for this goal. We have also introduced some performance problems and how we can overcome them. Finally, we have explained how to collect the required data to conver this framework in a real tool during the next year.

Acknowledgments. This work is partially supported by the Ministry of Science and Technology of Spain under contract TIN2012-34557 and by the BSC-CNS Severo Ochoa program (SEV-2011-00067) and with the support of ACC1Ó, the Catalan Agency to promote applied research and innovation; and by the Spanish Centre for Development of Industrial Technology under the INNPRONTA program, project IPT-20111006, "CIUDAD2020".

References

1. E. Achtert, H.P. Kriegel, and A. Zimek. Elki: A software system for evaluation of subspace clustering algorithms. In *20th Int. Conf. on Scientific and Statistical Database Management (SSDBM)*, volume 5069 of *Lecture Notes in Computer Sciences (LNCS)*, pages 580–585. Springer-Verlag, 2008.
2. R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. *SIGMOD Record*, 27(2):94–105, 1998.
3. Apache Software Foundation. Hadoop Distributed File System (HDFS) Architecture.
4. Hal Berghel and David Roach. An extension of Ukkonen's enhanced dynamic programming asm algorithm. *ACM Transactions Information Systems*, 14(1):94–106, 1996.
5. Edgar Chávez, Gonzalo Navarro, Ricardo Baeza-yates, and José L. Marroquín. Searching in metric spaces. *ACM Computing Surveys*, 33:273–321, 1999.
6. Thomas Cover and Peter Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967.
7. Yves-Alexandre de Montjoye, César A Hidalgo, Michel Verleysen, and Vincent D Blondel. Unique in the crowd: The privacy bounds of human mobility. *Scientific reports*, 3, 2013.
8. G. Dong and J. Pei. *Sequence Data Mining*. Springer, 2007.
9. M. Ester, H. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *2nd Int. Conf. on Knowledge Discovery and Data Mining*, pages 226–231, 1996.
10. Blake G. Fitch, Aleksandr Rayshubskiy, Michael C. Pitman, T. J. Christopher Ward, and Robert S. Germain. Using the active storage fabrics model to address petascale storage challenges. In *Proceedings of the 4th Annual Workshop on Petascale Data Storage*, PDSW '09, pages 47–54, New York, NY, USA, 2009. ACM.

11. Brad Fitzpatrick. Distributed caching with memcached. *Linux J.*, 2004(124):5–, August 2004.
12. Lorenzo Gabrielli, Salvatore Rinzivillo, Francesco Ronzano, and Daniel Villatoro. From tweets to semantic trajectories: mining anomalous urban mobility patterns. In *Proc. European Conf. on Complex Systems ECCS 2013*. Barcelona, Spain, Springer, 2013.
13. GeoNames. GeoNames geographical database, 2010. Last access on July 2013 at: http://www.geonames.org/.
14. Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system. *SIGOPS Oper. Syst. Rev.*, 37(5):29–43, 2003.
15. C. Gómez-Alonso and A. Valls. A similarity measure for sequences of categorical data based on the ordering of common elements. In *Proc. of the Int. Conf. of Modeling Decisions for Artificial Intelligence (MDAI)*, Lecture Notes on Artificial Intelligence, pages 134–145. Springer, 2008.
16. Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: an update. *ACM SIGKDD Explorations Newsletter*, 11(1):10–18, 2009.
17. R. W. Hamming. Error detecting and error correcting codes. *Bell System Technical Journal*, 26(2):147–160, 1950.
18. J. Herranz, J. Nin, and M. Solé. Optimal symbol alignment distance: A new distance for sequences of symbols. *IEEE Trans. on Knowledge and Data Engineering (TKDE)*, 23(10):1541–1554, 2011.
19. A. K. Jain, M. N. Murty, and P.J. Flynn. Data clustering: a review. *ACM Computing Surveys*, 31(3):264–323, 1999.
20. N. Jardine and R. Sibson. The construction of hierarchic and non-hierarchic classifications. *The Computer Journal*, 11(2):177–184, 1968.
21. M.A. Jaro. Advances in record-linkage methodology as applied to matching the 1985 census of tampa, florida. *Journal of the American Statistical Association*, 84:414–420, 1989.
22. Avinash Lakshman and Prashant Malik. Cassandra: a structured storage system on a p2p network. In *Proceedings of the twenty-first annual symposium on Parallelism in algorithms and architectures*, SPAA '09, pages 47–47, New York, NY, USA, 2009. ACM.
23. J.G. Lee, J. Han, X. Li, and H. Gonzalez. Traclass: Trajectory classification using hierarchical regionbased and trajectorybased clustering. In *ACM Very Large Data Base (VLDB)*, 2008.
24. J.G. Lee, J. Han, and K.Y. Whang. rajectory clustering algorithms. *Inter. Conf. Management of data (SIGMOD)*, pages 593–604, 2007.
25. V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(707–710), 1966.
26. S. LLoyd. Least squares quantization in pcm. *IEEE Trans. on Information Theory*, 28:129–137, 1982.
27. Geohash, http://docs.mongodb.org/manual/core/geospatial-indexes.
28. MongoBD, http://www.mongodb.org.
29. G. Navarro. A guided tour to approximate string matching. *ACM Computing Surveys*, 33(1):31–88, 2001.
30. Anastasios Noulas, Salvatore Scellato, Renaud Lambiotte, Massimiliano Pontil, and Cecilia Mascolo. A tale of many cities: universal patterns in human urban mobility. *PloS one*, 7(5):e37027, 2012.
31. E. Ristad and P. Yianilos. Learning string edit distance. *IEEE Transactions on Pattern Recognition and Machine Intelligence*, 20(5):522–532, 1998.

32. P. Selllers. The theory and computation of evolutionary distances: pattern recognition. *Journal of Algorithms*, pages 359–373, 1980.
33. A.K.H. Tung, J. Hou, L.V.S. Lakshmanan, and R.T. Ng. Constraint-based clustering in large databases. In *8th Int. Conf. on Database Theory*, pages 405–419, 2001.
34. Esko Ukkonen. On approximate string matching. In *Proc. of the Int. FCT-Conference on Fundamentals of Computation Theory*, pages 487–495, 1983.
35. Daniel Villatoro, Jetzabel Serna, Víctor Rodríguez, and Marc Torrent-Moreno. The tweetbeat of the city: Microblogging used for discovering behavioural patterns during the mwc2012. In *Citizen in Sensor Networks*, pages 43–56. Springer, 2013.
36. Robert A. Wagner and Michael J. Fischer. The string-to-string correction problem. *Journal of ACM*, 21(1):168–173, 1974.
37. X. Wang and H.J. Hamilton. Dbrs: A density- based spatial clustering method with random sampling. In *7th Pacific-Asia Conf. on Knowledge Discovery and Data Mining*, pages 563–575, 2003.
38. X. Wang and J. Wang. Using clustering methods in geospatial information systems. *Geomatica*, 64(3):347–361, 2010.
39. Yu Zheng, Lizhu Zhang, Xing Xie, and Wei-Ying Ma. Mining interesting locations and travel sequences from gps trajectories. In *Proceedings of the 18th international conference on World wide web*, pages 791–800. ACM, 2009.